RESEARCH-ARTICLE

Beyond Lamport, Towards Probabilistic Fair Ordering

MUHAMMAD HASEEB, New York University, New York, NY, United States

Preferred name: Haseeb Ashfaq

JINKUN GENG, Stanford University, Stanford, CA, United States

RADHIKA MITTAL, University of Illinois Urbana-Champaign, Urbana, IL, United States

AUROJIT PANDA, New York University, New York, NY, United States

SRINIVAS NARAYANA, Rutgers University–New Brunswick, New Brunswick, NJ, United States

ANIRUDH SIVARAMAN, New York University, New York, NY, United States

Open Access Support provided by:

Rutgers University–New Brunswick

New York University

Stanford University

University of Illinois Urbana-Champaign

Published: 17 November 2025

Citation in BibTeX format

HotNets '25: 24th ACM Workshop on Hot Topics in Networks
November 17 - 18, 2025
MD, College Park, USA

Conference Sponsors:
SIGCOMM

Beyond Lamport, Towards Probabilistic Fair Ordering

Muhammad Haseeb[n] Jinkun Geng[n][s] Radhika Mittal[u] Aurojit Panda[n]
Srinivas Narayana[r] Anirudh Sivaraman[n]
[n]*New York University* [s]*Stony Brook University* [r]*Rutgers University* [u]*UIUC*

ABSTRACT

A growing class of applications demands *fair ordering* of events, which ensures that events generated earlier are processed before later events. However, achieving such sequencing is challenging due to the inherent errors in clock synchronization: two events at two clients generated close together may have timestamps that cannot be compared confidently. We advocate for an approach that embraces, rather than eliminates, clock synchronization errors. Instead of attempting to remove the error from a timestamp, Tommy, our proposed system, leverages a statistical model to compare two noisy timestamps probabilistically by learning per-clock synchronization error distributions. Our preliminary statistical model computes the probability that one event precedes another by only relying on local clocks of clients. This serves as a foundation for a new relation: *likely-happened-before* denoted by $\xrightarrow{p}$ where p represents the probability that an event happened before another. The $\xrightarrow{p}$ relation provides a basis for ordering multiple events which are otherwise considered *concurrent* by Lamport's *happened-before* ($\rightarrow$) relation. We highlight various related challenges including the intransitivity of the $\xrightarrow{p}$ relation as opposed to the transitive $\rightarrow$ relation. We outline several research directions: online fair sequencing, stochastically fair total ordering, and handling byzantine clients.

CCS CONCEPTS

• **Networks** → **Application layer protocols**; • **Mathematics of computing** → **Probability and statistics**;

KEYWORDS

Fairness, Ordering, Sequencing, Clock Synchronization, Probabilistic Ordering

ACM Reference Format:
Muhammad Haseeb, Jinkun Geng, Radhika Mittal, Aurojit Panda, Srinivas Narayana, Anirudh Sivaraman. 2025. Beyond Lamport, Towards Probabilistic Fair Ordering. In *The 24th ACM Workshop on Hot Topics in Networks (HotNets '25), November 17–18, 2025, College Park, MD, USA.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3772356.3772401

HotNets '25, November 17–18, 2025, College Park, MD, USA

ACM ISBN 979-8-4007-2280-6/25/11…$15.00
https://doi.org/10.1145/3772356.3772401

1 INTRODUCTION

Sequencers play a pivotal role in distributed systems, providing a mechanism to impose a total order on events occuring potentially at different locations. They are essential components in several fundamental protocols, such as consensus and concurrency control. In consensus protocols (e.g., Paxos [1] and Raft [2]), the leader node serves as the sequencer for deciding a total order, as well as an orchestrator for achieving agreement on the total order. More recently, network-based sequencers have been introduced to offload some of the complexity from these protocols. Systems such as NOPaxos [3], Hydra [4], and Eris [5] decouple sequencing from the rest of the functionality, proposing dedicated sequencers to improve overall system efficiency.

At its core, the function of a sequencer is simple: assign ranks to incoming messages, thereby establishing a total order for processing the messages. This ranking is typically independent of when a message was originally generated. Instead, it is assigned based on the order in which it is *observed* by a server/sequencer (i.e., FIFO sequencer). In most traditional applications, this FIFO approach suffices, as the system only requires *some* ordering, even if arbitrary.

We make a case for fair ordering which, unlike FIFO ordering, requires that *an earlier generated event is ordered before a later generated event.* The FIFO order could be naturally closer to the fair order if the time between generation of every two events is large enough that arbitrary network delays do not obscure the order of events. However, there is a rise in applications in which a large volume of events is generated close together. These applications demand a sequencing mechanism that explicitly aligns the ordering of events with the timestamps at which the events are generated. It is particularly prominent in financial exchanges, ad

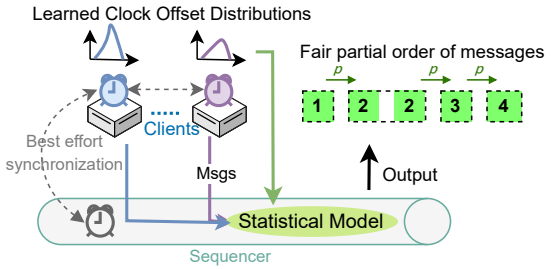

Figure 1: The sequencer, Tommy, uses clock offset distributions and noisy timestamps of messages to achieve a fair ordering of messages via a statistical model. Messages whose order cannot be confidently determined become part of the same output batch.

exchanges, and other competitive systems [6–12], where fairness is paramount, we call such applications *auction-apps*. In such applications, millions of events by hundreds of clients are generated within a very small window of time upon some sensitive event, for example, in financial exchanges some event leading to market volatility may be broadcasted to all the clients simultaneously [7, 8, 13], eliciting a large volume of responses by the clients. In these settings, ensuring that an earlier-generated message is ranked lower (processed sooner) than a later-generated one is crucial for maintaining fairness among participants. It is because of such fairness requirements and lack of fair ordering primitives, that exchanges today are built in private data-centers and not on a general purpose networking fabric e.g., that of the public cloud as seen by its tenants.

Recent Efforts: Recently the community has alluded to such ordering in the context of auction-apps, but either the solutions are (i) impractical [7, 13] due to strong assumptions (e.g., near-perfect clock synchronization), or (ii) not generally reusable because of being coupled with the intricacies of a particular application [8]. We define a general mechanism for achieving fair ordering as a *fair sequencer*: a sequencer that guarantees that an earlier event is ranked lower (i.e., processed sooner) than a later event with a high probability.

Classical Context: Lamport's seminal work on ordering of events [14] introduces the *happened-before* ($\rightarrow$) relationship. If two events a and b are causally related i.e., a causes b, then they can be ordered i.e., $a \rightarrow b$. The relation $\rightarrow$ is a transitive relation so a set of related events can be partially ordered. Two concurrent events, i.e., for whom a causal relationship cannot be determined are left unordered i.e., $a \nrightarrow b$ and $b \nrightarrow a$. We are precisely interested in ordering such concurrent events; a hard feat in its general essence as we establish in this paper, but very much needed for fair ordering.

Fundamental Challenge: Ideal fair ordering requires perfect clock synchronization so that two timestamped-events (from two different clients) can be ordered correctly even if the network reorders them. Perfect clock-synchronization is impossible to achieve in asynchronous or bounded-synchronous networks [15, 16] due to fundamental uncertainty around link delays. It is impossible to synchronize clocks of n processes any more closely than $u(1 - 1/n)$ where u represents the uncertainty in the link delays [16]. This impossibility of clock synchronization makes it challenging to achieve fair ordering even if all parties are trusted [8].

An Approximate Solution and When It Fails: In a constrained setting where the time resolution of interest is significantly coarser than the clock synchronization errors, the fair sequencer can be implemented by a straightforward algorithm as clock errors can be effectively ignored: by waiting for at least one message from every client and then releasing the message with the smallest timestamp, iteratively. This algorithm achieves a fair total order, provided in-order delivery of messages per client. This approach is practical in environments where all client VMs and the sequencer reside within a single data center, as clock synchronization errors can be reduced to nanoseconds [17], making it practical for systems operating at microsecond or higher time resolutions. However, when the required resolution is finer or clock synchronization errors become pronounced, such as in multi-data center deployments where the errors easily reach tens of microseconds, this approach is insufficient. To address these broader challenges, we call for a generally fair sequencer.

A Research Vision and Associated Challenges: We advocate leveraging the insight that two *local* timestamps from two clients can be compared if the clock offsets distributions of the clients are known. A client can learn its distribution of clock *offsets* (w.r.t. the sequencer's clock), for example, by accumulating synchronization probes[1] from any clock synchronization protocol. The learned offsets' distributions are shared with the sequencer, enabling a comparison of two local timestamps. Figure 1 sketches a possible system architecture. Based on this ability, we introduce a new relation: *likely-happened-before*, $\xrightarrow{p}$ where p denotes the probability i.e, in $x \xrightarrow{p} y$, x happened before y with probability p. Similar to how $\rightarrow$ relation is used for defining a partial order on events, the $\xrightarrow{p}$ relation can be used to provide a *fair* partial order. However, as the $\xrightarrow{p}$ relation is probabilistic, ordering *all* concurrent events with high confidence may not always be possible. Hence, only a partial order is expected. It is important to minimize such instances of *non-ordering* as otherwise a trivial solution is to leave all events as unordered. This ordering based on $\xrightarrow{p}$ constitutes fair ordering.

[1] A synchronization probe is a packet sent by a clock synchronization protocol from one client to the other to find and correct any clock offset.

There are two main challenges in using the $\xrightarrow{p}$ relation to achieve fair ordering: (i) unlike the $\rightarrow$ relation, the $\xrightarrow{p}$ relation is not necessarily transitive, so using it to order more than two events is non-trivial and, (ii) finding the probability p for constructing $\xrightarrow{p}$ relations. We later present a preliminary statistical model to calculate p. Once p is known, it can be used to obtain an ordering which has high confidence (§3.4).

Intransitivity and Ordering of Multiple Events: It is possible for the probability of event A preceding event B to be high, the probability of B preceding C to be high, and yet the probability of C preceding A to also be high. In a similar vein, an ordinary cat may prefer fish to meat, meat to milk and milk to fish, in exhibiting cyclic ordering. This renders $\xrightarrow{p}$ not necessarily a transitive relation, hindering us from defining an order on the events from pairwise relations. We later discuss a solution for handling such intransitivity, while also presenting a sequencer for the case where probabilities are transitive. Transitivity exists for some *nicely shaped* distributions like Gaussian distributions (proof in a tech report [18]) but may not hold for arbitrary distributions (e.g., [19]).

Furthermore, online sequencing is an equally challenging problem as sequencing a given set of events primarily because of (i) network asynchrony and, (ii) figuring out whether some future events may need the same or lower rank than some given events. We later discuss a direction for achieving online sequencing. We prototype our statistical approach, Tommy, and present simulation results demonstrating its effectiveness compared to a naive TrueTime (Spanner) based baseline [20]. We highlight a range of research directions enabled by our approach –potentially culminating in a novel sequencing primitive that supports a broad class of emerging applications atop general-purpose networking infrastructure.

2 RELATED WORK AND MOTIVATION

Cloud Exchanges: Recent proposals for cloud-hosted financial exchanges [7, 8, 13] deal with the same sequencing problem. However these systems either simplify the problem by making strong assumptions like negligible clock synchronization errors or they reduce the scope of the problem by limiting what kinds of events are possible. Figure 2 shows a **W**aits **F**or **O**ne (WFO) sequencer which waits for one message from all clients and releases the one with the smallest timestamp, iteratively. This sequencer is employed by [21] and works as long as the clock synchronization errors are small enough to be ignored so that the timestamps on the messages can be considered representing a global-clock time.

On-Prem Exchanges: On-premise exchanges engineer their infrastructure for fair ordering: connecting all clients to the server using equal length wires and employing low jitter

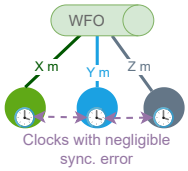

Figure 2: Fair if clocks are perfectly synchronized.

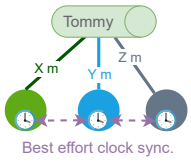

Figure 3: Fair w/o constraints but probabilistically.

Figure 4: Fair if all wires are of equal length.

switches (e.g., L1 switches [22]). In such a setting, the server can process messages in the order of their arrival which would be equivalent to ordering them on their generation timestamps (Figure 4). However, such a sequencer can only be deployed by modifying the underlying infrastructure. Tommy, our proposal, is a solution that does not make such assumptions or require special infrastructure (Figure 3).

Departing from Arbitrary Ordering: Pompe [23] proposes departing from an arbitrary total order and instead allowing the nodes of a Replicated State Machine to present hints about their desired ordering of events. However, Pompe is fundamentally different from Tommy in its goals. Pompe focuses on the question of how to keep a subset of replicas from influencing the ordering of events unilaterally. It has applications in settings e.g., blockchains, where Byzantine failures are possible. Tommy, on the other hand, focuses on whether and how an ordering of events can be achieved which reflects the true order of event occurrences.

Our Motivation: Our motivation stems from the efforts around migrating financial exchanges to the public cloud. Financial exchanges have traditionally been built in private data centers or colocation facilities, where the physical network is engineered to provide fairness guarantees. This eliminates the need for a fair sequencer in such environments. However, a recent wave of research [7, 8, 13, 21] exploring the migration of financial exchanges to the public cloud has created a demand for new networking primitives. One such primitive, briefly mentioned in Onyx [21], is a sequencer for fair total ordering. The design of Onyx assumes that clock synchronization errors are significantly smaller than the time resolution of interest, allowing it to disregard clock variability. However, we observe that this assumption does not hold if the system is deployed across multiple cloud regions where the clock synchronization errors can be significantly higher (e.g., in the order of milliseconds [24]), necessitating a more generalized fair sequencer.

Auction-apps: Beyond financial exchanges, many applications can benefit from such a sequencer, including ad exchanges and competitive marketplaces. *An application involving a shared state among multiple clients, where any particular*

order of writes can be advantageous/disadvantageous for some clients i.e., clients may compete to write earlier than others, is a candidate for fair sequencing. We call such applications *auction-apps*. The rise of competitive marketplaces [6–12] and our discussion with experts demonstrate that such applications are becoming ubiquitous.

Fairness: We use the term fairness differently from the typical networking/scheduling notions of fairness i.e., Jain's index [25], CFS [26] or throughput-centric fairness. We define fairness in sequencing as follows:

Definition 1 (Fair Sequencing)**.** Messages generated by the clients should be seen by a server in the same order as their generation is observed by an omniscient observer.[2]

Other notions of fairness in sequencing may also be possible, but we only focus on Definition 1. Furthermore, in practice, the timestamp (w.r.t the local clock) at which an event actually occurs and the timestamp (w.r.t the local clock) that is associated with the event generation by a client can have non-zero difference because of some *latency* between the application and the clock. For the sake of this position paper, we assume the difference is negligible.

3 PRELIMINARY DESIGN FOR TOMMY

Each client's clock may have some error w.r.t. the sequencer's clock due to imperfect clock synchronization.[3] The sequencer, Tommy, receives messages from clients with timestamps attached, attempts to order them and form batches ($B_i, B_j, ..$). All messages within a batch B_i will have a rank i where successive batches have higher ranks. Ideally, if message a is created before message b according to the global-clock, then the rank of the batch containing a should be smaller than the rank of the batch containing b. If two timestamps cannot be ordered confidently, then the corresponding messages should be part of the same batch. The challenge is to come up with the batches that maximizes fairness: the more batches we make, the better fairness we achieve.[4]

We decompose the above problem into two steps: (i) finding the probability of one message preceding another message (§3.2, §3.3) to construct the $\xrightarrow{p}$ relation and, (ii) using the pairwise relationships to get ordered batches (§3.4) that provides a fair partial order on all messages. We assume all messages are present at the sequencer before it starts sequencing. Later in §3.5, we lift this assumption. The preliminary system does not handle the case where clock offset distributions lead to intransitive probabilities on the order of events, but we discuss a direction for a possible solution.

3.1 System Model

Each client submits a message to the sequencer and attaches the current timestamp from its local clock. A message i has timestamp T_i. However, due to clock synchronization errors, the true timestamp of the message (from the sequencer's perspective) is: $T_i^* = T_i + \theta_i$ where θ_i represents the clock offset of a client (w.r.t the sequencer's clock) at the exact moment when the message i is generated. The offset θ_i is unknown but follows probability distribution f_{θ_i}. The sequencer can observe T_i, not T_i^*.

Different clients may have different distributions due to heterogeneous synchronization conditions (e.g., different temperature in different parts of a data center, asymmetric latency between clients). Each client learns their own distribution (by accumulating clock synchronization probes) and provides information about their distribution to the sequencer (§5).

3.2 Ordering Probability

It is impossible to compute T_i^* exactly but we can compare two timestamps T_i^* and T_j^* by only observing T_i and T_j using a probabilistic analysis that assumes the knowledge of clock offset distributions f_{θ_i} and f_{θ_j}.

We analyze the probability that one event/message precedes another. This probability is called the *preceding-probability*:

$$\mathbb{P}(T_i^* < T_j^* \mid T_i, T_j) = \mathbb{P}(T_i + \theta_i < T_j + \theta_j).$$

Rearranging,

$$\mathbb{P}(T_i^* < T_j^* \mid T_i, T_j) = \mathbb{P}(\theta_j - \theta_i > T_i - T_j).$$

Since θ_i and θ_j are random variables, their difference follows a new distribution:

$$\Delta\theta = \theta_j - \theta_i \sim f_{\Delta\theta}.$$

Then the preceding-probability is given by:

$$\mathbb{P}(T_i^* < T_j^* \mid T_i, T_j) = \int_{T_i - T_j}^{\infty} f_{\Delta\theta} d\Delta.$$

If two independent random variables follow Gaussian distributions, then their difference also follows a Gaussian distribution [27]. Therefore, for independent Gaussian-distributed clock synchronization errors, $\Delta\theta$ would be Gaussian-distributed. In this case, the preceding-probability is simply $\Phi\left(\frac{T_j - T_i + (\mu_i - \mu_j)}{\sqrt{\sigma_i^2 + \sigma_j^2}}\right)$, where $\Phi(x)$ is the standard normal CDF, and μ_i and σ_i^2 respectively represent the mean and variance of f_{θ_i}.

[2] An omniscient observer has access to a global clock with infinite resolution and has instantaneous knowledge of all events. It serves as an idealized scheme to compare against.

[3] Synchronizing clients' clocks with the sequencer's clock is sufficient as opposed to synchronizing clients' and sequencer's clock with a global clock.

[4] Assuming no two events occur at the same instant.

3.3 Handling Arbitrary Distributions

When the clock offsets θ_i and θ_j follow arbitrary distributions rather than Gaussian or when we are uncertain about the distribution of $\Delta\theta$, we may not have a well-known solution form. Such cases have been reported where although the clock-offsets data appear Gaussian-like, it shows a long tail and skewed behavior [28]. We must estimate the PDF $f_{\Delta\theta}$ for each pair of clients to compute the preceding probabilities to account for non-Gaussian behavior.

Computing all $\Delta\theta$s to get $f_{\Delta\theta}$: For each round of clock synchronization probes to the clients, a sequencer could gather *all* the probes and calculate pairwise probe differences ($\Delta\theta$s) and learn their distribution ($f_{\Delta\theta}$) across several rounds. This is communication and computation intensive. If a clock sync. protocol has a high probe frequency, it would increase the communication to sequencer as well. However, a simpler and efficient method exists, explained below.

Clients learn their own f_{θ_i}: If clients learn their own offset (w.r.t. the sequencer's clock) distributions over several rounds of clock synchronization, they can share their respective distributions with the sequencer which could perform (pairwise) convolutions to estimate $f_{\Delta\theta}$ for each pair of clients.

Convolution for finding the Probability Density Function (PDF): The PDF of $\Delta\theta = \theta_j - \theta_i$ is given by the *convolution* of the individual PDFs θ_i and θ_j i.e., $f_{\Delta\theta}(\Delta) = \int_{-\infty}^{\infty} f_{\theta_j}(\xi) f_{\theta_i}(\xi - \Delta) d\xi$. This approach requires less communication from the clients to the sequencer as clients merely send their respective learned distributions to the sequencer as opposed to sending all the clock synchronization probes.

Optimizing the convolutions calculations: The calculations of all pairwise convolutions at the sequencer can further be optimized by leveraging Fast Fourier Transform: convolution in the time domain is multiplication in the frequency domain. Instead of computing a convolution, we can (i) compute Fourier transforms of f_{θ_j} and $f_{-\theta_i}$, (ii) multiply them pointwise and, (iii) compute the inverse Fourier transform to get $f_{\Delta\theta}$. This process has log-linear time complexity if using FFT, as opposed to the quadratic complexity of convolution.

Once $f_{\Delta\theta}$ is obtained, the preceding-probability is simply: $\mathbb{P}(T_i^* < T_j^* \mid T_i, T_j) = \int_{T_i - T_j}^{\infty} f_{\Delta\theta}(\Delta) d\Delta$. This framework supports arbitrary clock error models, making it robust for real-world environments.

3.4 Fair Ordering

Once we define the $\xrightarrow{p}$ relation, we can work towards ordering multiple events. We model each message as a node in a graph, where $\xrightarrow{p}$ denotes a directed edge with weight p. In our construction, there will be two edges between each pair of nodes; for every such pair, we discard the edge with the lower weight (assuming no ties). From the resultant graph, we can extract a linear ordering of events by finding a topological ordering. Questions remain whether a topological ordering exists or which topological ordering to select if multiple orderings are possible.

Assuming clock offsets distributions that lead to transitivity for $\xrightarrow{p}$, the graph forms a *transitive tournament* [29].[5] Transitive tournaments have a unique Hamiltonian path, hence a unique topological ordering. So the problem simplifies in the case of transitivity. In a tech report [18], we prove how Gaussian distributions always lead to the required transitivity. The tech report [18] illustrates an example with several events and respective transitive preceding probabilities and how fair ordering is achieved.

In the case of intransitivity of $\xrightarrow{p}$, the resulting graph could be cyclic so no topological ordering may be possible. We may need some transformation of the graph to enable extracting a (most probable) linear ordering. One option is to remove some edges that renders the graph acyclic. However, it would lead to unfairness towards some messages/clients. For example, for three events whose preceding probabilities form a cycle, we could remove one edge to get a linear ordering but the removal of the edge would lead to ignoring one preceding-probability in the final linear ordering. A notion of stochastic fairness could be introduced and every time a set of messages is processed, we remove some edges from the graph in a fashion that leads to fairness over the long run. However, finding the smallest set of edges whose removal would make a graph acyclic is an NP-hard problem. These aspects of fair ordering make the problem non-trivial under intransitivity, warranting further research.

The extracted linear ordering from the graph, even under transitive probabilities, cannot be construed as a final ordering. $\xrightarrow{p}$ relations of some adjacent messages in the linear ordering have a p just slightly above 0.5 while other may have a p close to 1; so it cannot be considered fair with a reasonable confidence. We batch adjacent messages such that if $i \xrightarrow{p} j$ has $p > threshold$ then a batch boundary is created between i and j, making i and j belong to two different batches. Finally, the first such batch is assigned a rank of 0 while successive batches get incremental ranks, yielding a fair ordering of messages. The messages which we cannot order confidently become part of the same batch; thus our ordering is partial and not total. The hyper-parameter

[5] A directed graph with exactly one edge between every pair of nodes is called a tournament. A tournament in which the edge relation is transitive is called a transitive tournament.

Threshold dictates the confidence of our ordering and needs to be selected carefully.

A *Threshold* closer to 1 creates fewer and bigger batches, while a *Threshold* closer to 0.5 creates smaller and more batches. A higher value of *Threshold* provides more confidence in the output ordering but may lead to more number of messages left as unordered i.e., part of the same batch. Ideally, each batch should be of size 1. Hence, maximizing fairness amounts to creating smaller batches. While maximizing correctness may require staying indifferent about the (concurrent) messages, i.e., making them part of the same batch as we can never be 100% confident about ranking of batches. We leave the optimization of *Threshold* as future work and currently use a value of 0.75 in the evaluation.

Although we achieve partial ordering on the messages, it is a total ordering on the batches. The sequencer emits one batch at a time to an upstream application for further processing of the corresponding messages.

3.5 Online Sequencing

The above discussion on ordering assumes that the sequencer has received all the messages that need to be sequenced. However, in practice, messages arrive as a stream, and the sequencer must operate in an online fashion. Crucially, the sequencer must ensure that once a batch of messages is *emitted*, i.e., released after sequencing, no new message should arrive that either belongs in the same batch or demands a lower rank.

Two main questions: The challenge of online sequencing boils down to answering two key questions. **Q1:** Given a batch of timestamps (of messages), what future timestamps might still need to be included in the current batch? **Q2:** How can we ensure that all messages with timestamp t (or $\leq t$) have already arrived at the sequencer? Q1 arises due to clock synchronization errors –specifically, a client c may have enough uncertainty in their local timestamps that messages from another client, with later timestamps, must be grouped with c's messages. In such scenarios, although two messages i, j from a client can be ordered w.r.t each other, they must belong to the same batch as a third *high-uncertainty message* k from another client. This is required because $\mathbb{P}(T_i^* < T_k^*)$ as well as $\mathbb{P}(T_j^* < T_k^*)$ can both be very small. The second question reflects the challenges introduced by network asynchrony. Our accompanying tech report [18] illustrates online sequencing with an example.

Safe batch emission: We hint at how the answer to Q1 can be extracted which is equivalent to calculating *waiting-period* to safely emit a batch. The sequencer can safely emit a batch if no new message that needs a lower or equal rank arrives during this waiting period, otherwise a new waiting period is calculated accounting for the newly received messages.

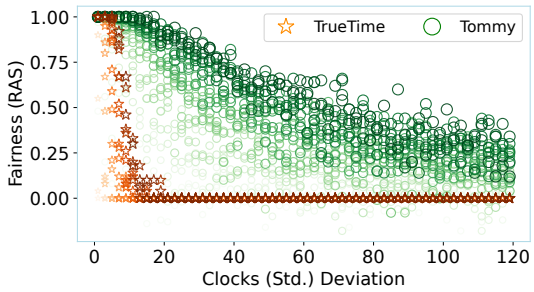

Figure 5: Tommy achieves fairer sequencing than TrueTime. Size of the marker (and color intensity) is proportional to the inter-messages gap across clients.

This could in theory lead to blocking the sequencer from emitting any messages if the arrival pattern of messages and the clock offsets distributions are set adversely. We have not tackled this yet.

A safe way to emit a batch is to calculate a future time T_i^F for each message i in the batch such that

$$\mathbb{P}(T_i^* < T_i^F) > p_{\text{safe}}$$

where p_{safe} can be set to a high value to ensure enough confidence (e.g., 0.999). T_i^F that respects the above constraint can be trivially and efficiently computed by a binary search on the future timestamps.

The safe emission time for the entire batch becomes:

$$T_b = \max_k \left(T_k^F\right) \quad \forall k \in \text{batch}$$

The sequencer after finalizing a batch, will only emit it (i) once its clock reaches T_b timestamp and, (ii) it has not received any further messages that should be part of the batch or deserve a lower rank. If new messages arrive before T_b which violate (ii), then T_b is extended accounting for the new messages. The parameter p_{safe} presents a trade-off between latency of emitting a batch and certainty of fairness.

Dealing with network asynchrony: There are several directions for dealing with network asynchrony (for Q2). Assuming bounded asynchrony and waiting for sufficiently long enough is a common practice while [7] studies the impact of waiting period on fairness. Another direction, applicable to *auction-apps* is to assume the knowledge of a fixed number of clients. This simple knowledge is powerful in answering Q2. To ensure all messages generated before some timestamp t have arrived, the sequencer simply waits for messages or heartbeats with timestamp greater than t from *all clients*. This works as long as the communication between each client and the sequencer happens through an ordered delivery channel (e.g., TCP connection). However, this design may cost liveness i.e., a failed client may halt the sequencer from emitting any messages.

4 EVALUATION

We evaluate our statistical model using a simulator with 500 clients, each assigned a Gaussian clock offsets distribution, $N(\mu, \sigma^2)$. At message generation, a client reads the wall-clock time t, samples noise ϵ from the distribution, and tags the message with $T = t + \epsilon$. The sequencer receives all messages before ordering, i.e., we do not evaluate online sequencing. Ground-truth timestamps (t) are also collected for evaluation. Clients send their distributions and timestamped messages to the sequencer as shown in Figure 1. We seed the clients with clock offsets distributions, instead of clients learning such distributions, so the following results are an upper-bound on the performance as the errors in estimating such distributions are not captured.

For baseline, we emulate Spanner TrueTime [20], where each message is assigned an uncertainty interval $[T-3\sigma,\ T+3\sigma]$, and overlapping intervals are assigned the same rank.

We propose a metric, Rank Agreement Score (RAS): +1 for each correct ordered pair, −1 for incorrect, and 0 for indifference i.e., for assigning same batch to a pair of messages.

Figure 5 shows RAS (each point is the sum of RAS of all pairs of messages) for both approaches, with marker size (and color intensity) showing inter-messages gaps across clients. With low clock errors (lower x-axis), both systems perform comparably. Tommy outperforms (higher y-axis) TrueTime, when inter-messages gap decreases (marker size/color intensity decreases) and/or clock errors increase (higher x-axis). However, Tommy's probabilistic nature can lead to negative RAS under high uncertainty/high clock errors, whereas TrueTime's RAS remains 0 due to its conservative nature.

5 FUTURE RESEARCH

Characterization of $\xrightarrow{p}$: Unlike Lamport's $\rightarrow$ relation, $\xrightarrow{p}$ relation is not necessarily transitive, which makes extracting the linear ordering a challenge. More research is needed to (i) render $\xrightarrow{p}$ transitive by some transformation of the problem space (e.g., barring the relation of some elements by enforcing constraints on event occurrence pattern), and (ii) studying the probability distributions of clock offsets to establish when $\xrightarrow{p}$ can be safely treated as transitive.

Host-network variability: Jitter in the host's data path can affect an application's access to the local clock as well as the latency of sending out a message. Advancements in low-latency and low-jitter host networking (e.g., DPDK [30], XDP [31], RTOS [32]) have minimized latency variations in the host data path. However, it remains to be studied how low latency variance can be reduced and whether it sets an upper bound on the achievable fairness guarantees.

Extension to Fair Total Order: The proposed sequencer emits batches instead of individual messages. As the batch size can be arbitrarily large, some applications may require emitting individual messages instead of batches. Doing this would require extending the fair partial order to fair total order of messages. Arbitrarily breaking ties on messages of a batch would violate fairness as some clients may always be preferred over others. A random mechanism for breaking ties might be of interest as it would lead to stochastic fairness over a sufficiently long duration.

Learning Clock Offsets Distributions: Any clock synchronization protocol gives each client enough information to estimate its offsets distribution. Each synchronization probe may add an offset (w.r.t. to the sequencer's clock) to the clock of a client. Such offsets can be used to estimate the distribution. This mechanism may be too brittle for extraordinary conditions like a part of the data-center experiencing abrupt temperature changes, leading to dramatic clock sync. errors. A robust mechanism for capturing such errors in the respective distributions is needed. Similarly, more research is needed to account for the clock drift errors along with the clock offsets errors in the error distributions.

Byzantine Clients: Byzantine failures further complicate the problem of fair sequencing. A study about achievable fairness guarantees in the presence of Byzantine failures is needed. Pompe [23] can serve as a promising starting point for further exploration. In *auction-apps*, clients have an incentive to dictate sequencing of messages e.g., by manipulating the timestamps attached to the messages, as it may translate to monetary benefits e.g., winning trades in a financial exchange. In-depth investigation of security boundaries is needed to make fair sequencing practical. The trust models in [21] provide a starting point.

6 CONCLUSION

We present the problem of fair sequencing and associated challenges which warrant substantial future research. We advocate for utilizing clock offset distributions along with a best effort clock synchronization protocol to construct a pairwise relation, *likely-happened-before*. The proposed relation helps in achieving probabilistic fair ordering of events, useful for an emerging class of applications which require fairly ordering several concurrent events.

7 ACKNOWLEDGMENTS

We thank the reviewers and our shepherd, Jon Crowcroft, for their helpful comments. We thank Ramakrishnan Krishnamurthy and Fabian Ruffy for their helpful comments and productive brainstorming sessions. This work was supported by NSF CAREER award (2340748).

REFERENCES

[1] Leslie Lamport. The part-time parliament. *ACM Trans. Comput. Syst.*, 16(2):133–169, May 1998. ISSN 0734-2071. doi: 10.1145/279227.279229. URL https://doi.org/10.1145/279227.279229.

[2] Diego Ongaro and John Ousterhout. In search of an understandable consensus algorithm. In *Proceedings of the 2014 USENIX Conference on USENIX Annual Technical Conference*, USENIX ATC'14, page 305–320, USA, 2014. USENIX Association. ISBN 9781931971102.

[3] Jialin Li, Ellis Michael, Naveen Kr. Sharma, Adriana Szekeres, and Dan R. K. Ports. Just say NO to paxos overhead: Replacing consensus with network ordering. In *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*, pages 467–483, Savannah, GA, November 2016. USENIX Association. ISBN 978-1-931971-33-1. URL https://www.usenix.org/conference/osdi16/technical-sessions/presentation/li.

[4] Inho Choi, Ellis Michael, Yunfan Li, Dan R. K. Ports, and Jialin Li. Hydra: Serialization-Free network ordering for strongly consistent distributed applications. In *20th USENIX Symposium on Networked Systems Design and Implementation (NSDI 23)*, pages 293–320, Boston, MA, April 2023. USENIX Association. ISBN 978-1-939133-33-5. URL https://www.usenix.org/conference/nsdi23/presentation/choi.

[5] Jialin Li, Ellis Michael, and Dan R. K. Ports. Eris: Coordination-free consistent transactions using in-network concurrency control. In *Proceedings of the 26th Symposium on Operating Systems Principles*, SOSP '17, page 104–120, New York, NY, USA, 2017. Association for Computing Machinery. ISBN 9781450350853. doi: 10.1145/3132747.3132751. URL https://doi.org/10.1145/3132747.3132751.

[6] Amazon Ads. What is real-time bidding (rtb)? definition and importance. https://advertising.amazon.com/library/guides/real-time-bidding. Accessed: 2025-04-07.

[7] Ahmad Ghalayini, Jinkun Geng, Vighnesh Sachidananda, Vinay Sriram, Yilong Geng, Balaji Prabhakar, Mendel Rosenblum, and Anirudh Sivaraman. Cloudex: A fair-access financial exchange in the cloud. In *Proceedings of the Workshop on Hot Topics in Operating Systems*, HotOS '21, page 96–103, New York, NY, USA, 2021. Association for Computing Machinery. ISBN 9781450384384. doi: 10.1145/3458336.3465278. URL https://doi.org/10.1145/3458336.3465278.

[8] Eashan Gupta, Prateesh Goyal, Ilias Marinos, Chenxingyu Zhao, Radhika Mittal, and Ranveer Chandra. Dbo: Fairness for cloud-hosted financial exchanges. In *Proceedings of the ACM SIGCOMM 2023 Conference*, ACM SIGCOMM '23, page 550–563, New York, NY, USA, 2023. Association for Computing Machinery. ISBN 9798400702365. doi: 10.1145/3603269.3604871. URL https://doi.org/10.1145/3603269.3604871.

[9] Nike Shoe Bot. Nike shoe bot - the ultimate sneaker bot, 2025. URL https://www.nikeshoebot.com/. Accessed: 2025-03-19.

[10] AIO Bot. Aio bot | the ultimate sneaker bot for automatic copping, 2025. URL https://www.aiobot.com/. Accessed: 2025-03-19.

[11] Kasada. Nft bots: How they work and how to stop them, 2025. URL https://www.kasada.io/nft-bots/. Accessed: 2025-03-19.

[12] Arjun Balasingam, Karthik Gopalakrishnan, Radhika Mittal, Mohammad Alizadeh, Hamsa Balakrishnan, and Hari Balakrishnan. Toward a marketplace for aerial computing. In *Proceedings of the 7th Workshop on Micro Aerial Vehicle Networks, Systems, and Applications*, Dronet '21, page 1–6, New York, NY, USA, 2021. Association for Computing Machinery. ISBN 9781450385992. doi: 10.1145/3469259.3470485. URL https://doi.org/10.1145/3469259.3470485.

[13] Muhammad Haseeb, Jinkun Geng, Ulysses Butler, Xiyu Hao, Daniel Duclos-Cavalcanti, and Anirudh Sivaraman. Poster: Jasper, a scalable and fair multicast for financial exchanges in the cloud. In *Proceedings of the ACM SIGCOMM 2024 Conference: Posters and Demos*, ACM SIGCOMM Posters and Demos '24, page 36–38, New York, NY, USA, 2024. Association for Computing Machinery. ISBN 9798400707179. doi: 10.1145/3672202.3673728. URL https://doi.org/10.1145/3672202.3673728.

[14] Leslie Lamport. Time, clocks, and the ordering of events in a distributed system. *Commun. ACM*, 21(7):558–565, July 1978. ISSN 0001-0782. doi: 10.1145/359545.359563. URL https://doi.org/10.1145/359545.359563.

[15] Nikolaos M. Freris, Scott R. Graham, and P. R. Kumar. Fundamental limits on synchronizing clocks over networks. *IEEE Transactions on Automatic Control*, 56(6):1352–1364, 2011. doi: 10.1109/TAC.2010.2089210.

[16] Jennifer Lundelius and Nancy Lynch. An upper and lower bound for clock synchronization. *Information and Control*, 62(2):190–204, 1984. ISSN 0019-9958. doi: https://doi.org/10.1016/S0019-9958(84)80033-9. URL https://www.sciencedirect.com/science/article/pii/S0019995884800339.

[17] Yilong Geng, Shiyu Liu, Zi Yin, Ashish Naik, Balaji Prabhakar, Mendel Rosenblum, and Amin Vahdat. Exploiting a natural network effect for scalable, fine-grained clock synchronization. In *15th USENIX Symposium on Networked Systems Design and Implementation (NSDI 18)*, pages 81–94, Renton, WA, April 2018. USENIX Association. ISBN 978-1-939133-01-4. URL https://www.usenix.org/conference/nsdi18/presentation/geng.

[18] Muhammad Haseeb, Jinkun Geng, Radhika Mittal, Aurojit Panda, Srinivas Narayana, and Anirudh Sivaraman. Fair ordering, 2025. URL https://arxiv.org/abs/2510.13664.

[19] Richard P. Savage Jr. and. The paradox of nontransitive dice. *The American Mathematical Monthly*, 101(5):429–436, 1994. doi: 10.1080/00029890.1994.11996968. URL https://doi.org/10.1080/00029890.1994.11996968.

[20] James C. Corbett, Jeffrey Dean, Michael Epstein, Andrew Fikes, Christopher Frost, J. J. Furman, Sanjay Ghemawat, Andrey Gubarev, Christopher Heiser, Peter Hochschild, Wilson Hsieh, Sebastian Kanthak, Eugene Kogan, Hongyi Li, Alexander Lloyd, Sergey Melnik, David Mwaura, David Nagle, Sean Quinlan, Rajesh Rao, Lindsay Rolig, Yasushi Saito, Michal Szymaniak, Christopher Taylor, Ruth Wang, and Dale Woodford. Spanner: Google's globally distributed database. *ACM Trans. Comput. Syst.*, 31(3), August 2013. ISSN 0734-2071. doi: 10.1145/2491245. URL https://doi.org/10.1145/2491245.

[21] Muhammad Haseeb, Jinkun Geng, Daniel Duclos-Cavalcanti, Xiyu Hao, Ulysses Butler, Radhika Mittal, Srinivas Narayana, and Anirudh Sivaraman. Network support for scalable and high performance cloud exchanges. In *Proceedings of the ACM SIGCOMM 2025 Conference*, SIGCOMM '25, page 1110–1131, New York, NY, USA, 2025. Association for Computing Machinery. ISBN 9798400715242. doi: 10.1145/3718958.3750530. URL https://doi.org/10.1145/3718958.3750530.

[22] leptonsys.com. Layer 1 switches: Key functions and technologies. https://www.leptonsys.com/blog/layer-1-switches-key-functions-and-technologies.

[23] Yunhao Zhang, Srinath Setty, Qi Chen, Lidong Zhou, and Lorenzo Alvisi. Byzantine ordered consensus without byzantine oligarchy. In *14th USENIX Symposium on Operating Systems Design and Implementation (OSDI 20)*, pages 633–649. USENIX Association, November 2020. ISBN 978-1-939133-19-9. URL https://www.usenix.org/conference/osdi20/presentation/zhang-yunhao.

[24] Jinkun Geng, Shuai Mu, Anirudh Sivaraman, and Balaji Prabhakar. Tiga: Accelerating geo-distributed transactions with synchronized clocks. In *Proceedings of the ACM SIGOPS 31st Symposium on Operating Systems Principles*, SOSP '25, page 555–571, New York, NY, USA, 2025. Association for Computing Machinery. ISBN 9798400718700. doi: 10.1145/3731569.3764854. URL https://doi.org/10.1145/3731569.3764854.

[25] Raj Jain, Dah Ming Chiu, and Hawe WR. A quantitative measure of fairness and discrimination for resource allocation in shared computer

systems. *CoRR*, cs.NI/9809099, 01 1998.

[26] Chee Siang Wong, Ian Tan, Rosalind Deena Kumari, and Fun Wey. Towards achieving fairness in the linux scheduler. *SIGOPS Oper. Syst. Rev.*, 42(5):34–43, July 2008. ISSN 0163-5980. doi: 10.1145/1400097.1400102. URL https://doi.org/10.1145/1400097.1400102.

[27] John G. Proakis. Probability, random variables and stochastic processes. *IEEE Trans. Acoust. Speech Signal Process.*, 33:1637, 1985. URL https://api.semanticscholar.org/CorpusID:2072334.

[28] Ha Yang Kim. *Modeling and tracking time-varying clock drifts in wireless networks.* PhD thesis, Georgia Institute of Technology, Atlanta, GA, USA, 2015.

[29] S I Gass and. Tournaments, transitivity and pairwise comparison matrices. *Journal of the Operational Research Society*, 49(6):616–624, 1998. doi: 10.1057/palgrave.jors.2600572. URL https://doi.org/10.1057/palgrave.jors.2600572.

[30] DPDK Project. Data plane development kit (dpdk). https://www.dpdk.org/. Accessed: 2025-04-07.

[31] Toke Høiland-Jørgensen, Jesper Dangaard Brouer, Daniel Borkmann, John Fastabend, Tom Herbert, David Ahern, and David Miller. The express data path: fast programmable packet processing in the operating system kernel. In *Proceedings of the 14th International Conference on Emerging Networking EXperiments and Technologies*, CoNEXT '18, page 54–66, New York, NY, USA, 2018. Association for Computing Machinery. ISBN 9781450360807. doi: 10.1145/3281411.3281443. URL https://doi.org/10.1145/3281411.3281443.

[32] Real-Time Linux Project. Real-time linux. https://wiki.linuxfoundation.org/realtime/start. Accessed: 2025-04-07.